Personalized Recommendation System Using Collaborative Filtering

Technical Report

May 17, 2025

Abstract

The report describes a recommendation system that uses collaborative filtering to generate personalized product suggestions for retail customers. The model analyzes customer purchase patterns through Truncated SVD (Singular Value Decomposition) matrix factorization to identify product affinities. The system generates personalized suggestions from customer purchase records. The evaluation results demonstrate strong performance through RMSE of 0.3620 and MAE of 0.0884 which indicates the model successfully identifies customer preferences. The system operates with sample customer data from Coles supermarket after successful implementation and testing.

1 Data Overview

The model utilizes customer transaction data containing the following key information:

- **CustomerID**: Unique identifier for each customer
- **ProductCode**: Unique identifier for each product
- **ItemName**: Product description/name
- **Quantity**: Number of units purchased

Data preprocessing steps include:

- Creation of a user-item matrix where rows represent customers and columns represent products
- Values in the matrix represent purchase quantities
- Conversion to sparse matrix representation for memory efficiency
- Mapping of product codes to product names for interpretable recommendations

2 Methodology

2.1 Collaborative Filtering Approach

The system implements model-based collaborative filtering using matrix factorization, specifically:

2.1.1 Matrix Construction

- Creating a sparse user-item matrix from transaction data
- Each cell represents the quantity of a product purchased by a customer

2.1.2 Dimensionality Reduction

- Applying Truncated SVD to reduce the high-dimensional sparse matrix
- Using 40 latent factors to capture underlying purchase patterns
- This approach helps address the sparsity problem common in retail datasets

2.1.3 Similarity Computation

- Calculating cosine similarity between products in the reduced latent space
- Creating an item-item similarity matrix that captures product affinities

2.2 Recommendation Methods

The system implements one primary recommendation approaches:

2.2.1 Personalized Customer Recommendations

- Analyzes a customer's purchase history
- Identifies products with high similarity to previously purchased items
- Excludes already purchased products
- Ranks recommendations by aggregated similarity scores

3 Implementation Details

The implementation consists of four main components:

3.1 Data Loading and Preprocessing

```
def load_data(file_path):
    df = pd.read_csv(file_path)
    product_map = df[['ProductCode', 'ItemName']].drop_duplicates().
        set_index('ProductCode')['ItemName'].to_dict()

    user_item_matrix = df.pivot_table(
        index='CustomerID',
        columns='ProductCode',
        values='Quantity',
        aggfunc='sum',
        fill_value=0
    ).astype(np.float32)

    return csr_matrix(user_item_matrix.values), user_item_matrix.index,
        user_item_matrix.columns, product_map
```

3.2 Model Training

```
def train_model(matrix, components=40):
    svd = TruncatedSVD(n_components=components, random_state=42)
    item_factors = svd.fit_transform(matrix.T)  # Items in rows
    return svd, item_factors

def compute_similarities(item_factors):
    return cosine_similarity(item_factors)
```

3.3 Recommendation Functions

```
def personalized_recommendations(customer_id, user_index, sparse_matrix,
                                 similarity_matrix, items, product_map,
                                    top_n=10):
    try:
        customer_idx = list(user_index).index(customer_id)
        purchased_indices = sparse_matrix[customer_idx].indices
        scores = similarity_matrix[purchased_indices].sum(axis=0)
        scores[purchased_indices] = -np.inf  # Exclude purchased

        top_indices = np.argsort(-scores)[:top_n]
        return [(items[i], product_map.get(items[i], "Unknown"), scores[
            i])
                for i in top_indices if scores[i] > -np.inf]
    except ValueError:
        return []

def basket_recommendations(basket_items, similarity_matrix, items,
                           product_map, top_n=5):
    """Get recommendations based on current basket items"""
    try:
        # Convert basket items to indices (handle both string and
            integer codes)
        basket_indices = []
        for item in basket_items:
            item_str = str(item)  # Ensure string comparison
            if item_str in items:
                basket_indices.append(list(items).index(item_str))

        if not basket_indices:
            print("No valid basket items found in product catalog")
            return []

        # Aggregate similarities from all basket items
        scores = similarity_matrix[basket_indices].sum(axis=0)

        # Create mask for purchased items
        purchased_mask = np.zeros_like(scores, dtype=bool)
        purchased_mask[basket_indices] = True

        # Get top recommendations excluding basket items
        valid_scores = scores[~purchased_mask]
        valid_indices = np.argsort(-valid_scores)[:top_n]

        # Map back to original indices
        all_indices = np.arange(len(items))
```

```
        valid_global_indices = all_indices[~purchased_mask][
            valid_indices]

        return [
            (items[i], product_map.get(items[i], "Unknown Product"),
                scores[i])
            for i in valid_global_indices if scores[i] > 0
        ]
    except Exception as e:
        print(f"Error generating recommendations: {str(e)}")
        return []
```

3.4 Evaluation

```
# Train-test split
train, test = train_test_split(user_item_matrix, test_size=0.2,
    random_state=42)

# Fit model on train data
svd = TruncatedSVD(n_components=40, random_state=42)
svd.fit(train)
predicted = svd.inverse_transform(svd.transform(test))

# Calculate error metrics
rmse = np.sqrt(mean_squared_error(test, predicted))
mae = mean_absolute_error(test, predicted)
```

4 Results and Evaluation

4.1 Model Performance

The model evaluation using train-test split yielded the following metrics:

- **RMSE (Root Mean Square Error)**: 0.3620
- **MAE (Mean Absolute Error)**: 0.0884

These relatively low error values indicate that the model can effectively predict customer preferences and purchase patterns.

4.2 Sample Recommendations

Example personalized recommendations for customer C1001:

Rank	Product	Code	Relevance Score
1	Coles RSPCA Approved Chicken Mixed Portio ... 1.2kg	7382540	99.142
2	Coles Made Easy Slow Cooked Pork Shoulder | 480g	6413670	97.975
3	Coles Chocolate Mud Cake Slices 5 pack | 500g	3003661	97.792
4	Air Wick Essential Mist Revitalise Bamboo-Ber ... 20mL	4883436	97.614
5	Mission Souvlaki Bread | 320g	4327240	97.603

Table 1: Top 5 Personalized Recommendations for Customer C1001

The high relevance scores indicate strong affinities between these recommended products and the customer's previous purchases.

4.3 Observations

1. **Product Affinities**: The recommendations reveal logical product associations and complementary items (e.g., chicken products, prepared meals).
2. **Diverse Recommendations**: The system recommends products across different categories (meat, bakery, household items), indicating the model captures diverse shopping patterns.
3. **Basket Analysis**: The basket-based recommendation function provides contextual suggestions but requires valid product codes from the inventory to work properly.

5 Limitations and Challenges

1. **Cold Start Problem**:
   - New customers without purchase history cannot receive personalized recommendations
   - New products without purchase data cannot be effectively recommended
2. **Data Sparsity**:
   - Retail transactions typically create very sparse matrices
   - SVD helps address this, but extremely sparse data still presents challenges
3. **Memory Requirements**:
   - Computing full similarity matrices for large product catalogs requires significant memory
   - The implementation may need optimization for very large inventories

6 Conclusion

The implemented collaborative filtering system demonstrates strong potential for providing personalized product recommendations in a retail environment. The matrix factorization approach using Truncated SVD effectively captures customer preferences and product affinities, as evidenced by the low error metrics and relevant sample recommendations.

With the suggested improvements and enhancements, this system could become a valuable tool for improving customer experience, increasing basket size, and driving business growth through personalized marketing and recommendations.